EE2703 : Applied Programming Lab
Assignment 6
Laplace Tranforms

Sai Shashank GP
EE20B040

March 27, 2022

0.1 Introduction

In this assignment we will be dealing with transfer functions and analysis in frequency domain. We will also be using Bode plots to analyse phase and magnitude plots.
Libraries like **pylab** and **numpy** are used for visualisation and calculations. **scipy.signal** is a module in which we can find many functions related to transfer functions and manipulating them.
We will also be looking into *polynomial* objects in numpy.

0.2 Assignment

0.2.1 Prerequisites

We will be importing the aforementioned libraries

```
from pylab import *
import numpy as np
import scipy.signal as sp
```

And we will also define some functions which will come in handy later

```
def f(t, a:float=0.5, w:float=1.5):
    return np.where(t<=0, 0, np.cos(w*t)*np.exp(-1*a*t))
```

$$f(t) = cos(\omega t)e^{-\alpha t}u_0(t) \tag{1}$$

```
def vi(t):
    return np.where(t <= 0, 0, np.cos(10**3 * t) - np.cos(10**6 * t))
```

$$v_i(t) = cos(10^3 t)u(t) - cos(10^6 t)u(t) \tag{2}$$

0.2.2 Q1

In Q1, he gave us a function f(t) with it's laplace transform. He gave a second order differential equation which has f(t) as forced response. So, solving this in laplace is as follows.

```
def Q1(a:float=0.5, w:float=1.5, init_x:float=0.0, init_xdot:float=0.0):
    '''
    This function solves the question 1
    '''

    # Defining the laplace transform of function f(t) viz F
    global F_num,  F_den, F

    F_num = poly1d([1, a])
    F_den = poly1d([1, 2*a, w**2+a**2])
```

```
    F = sp.lti(F_num, F_den)

    # Defining the differential equation
    # x‘‘ + 2.25x = f(t) in time domain
    # s^2 . X - s . x‘(0) - x(0) - 2.25 . X = F in laplace domain

    global X_num, X_den, X

    X_num = polyadd(F_num, polymul(F_den, [init_x, init_xdot]))
    X_den = polymul(F_den, [1, 0, w**2])

    X = sp.lti(X_num, X_den)

    _, x = sp.impulse(X, None, t)

    # Plot the obtained x(t) from 0 to 50 seconds
    figure(0)
    plot(t, x)
    xlabel(r’time $\longrightarrow$’)
    ylabel(r’x(t)$\longrightarrow $ ’)
    title(r’solution of equation $\ddot x $ + 2.25x = f(t)’)
    show()
```

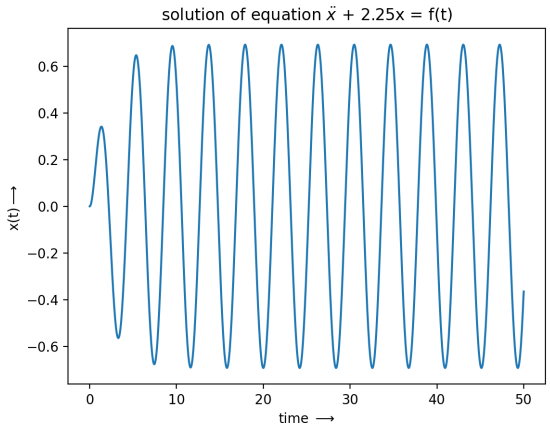

Figure 1: solution x(t) with $\alpha = 0.5$

0.2.3 Q2

Now, let’s change the decay coefficient to 0.05 and solve the differential equation

```
def Q2():
    ’’’
    This function solves the question 2
    ’’’
    Q1(a=0.05)
```

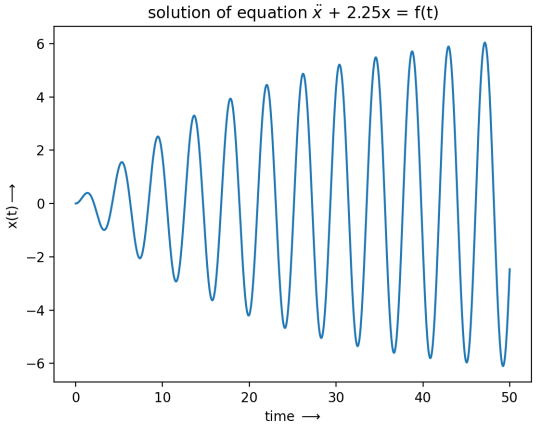

Figure 2: solution x(t) with $\alpha = 0.05$

0.2.4 Observation

When we decrease the decay parameter, α, the amplitude at which the function stabilises is increasing. This indicates that α controls the steady state amplitude.

0.2.5 Q3

Now, let's change the frequency of the cosine term in f(t).

```
def Q3():
    '''
    This function solves the question 3
    '''

    iterArr = np.arange(1.4, 1.6, 0.05)

    # Finding the transfer function X/F

    H_num = polymul(X_num, F_den)
    H_den = polymul(X_den, F_num)

    H = sp.lti(H_num, H_den)

    # Starting the loop and plotting

    t_ = np.linspace(0, 100, 2*N)

    for i in range(5):
        f_ = f(t_, a=0.05, w=iterArr[i])
        _, x, _ = sp.lsim(H, f_, t_)
        subplot(3, 2, i+1)
        plot(t_, x)

    show()
```

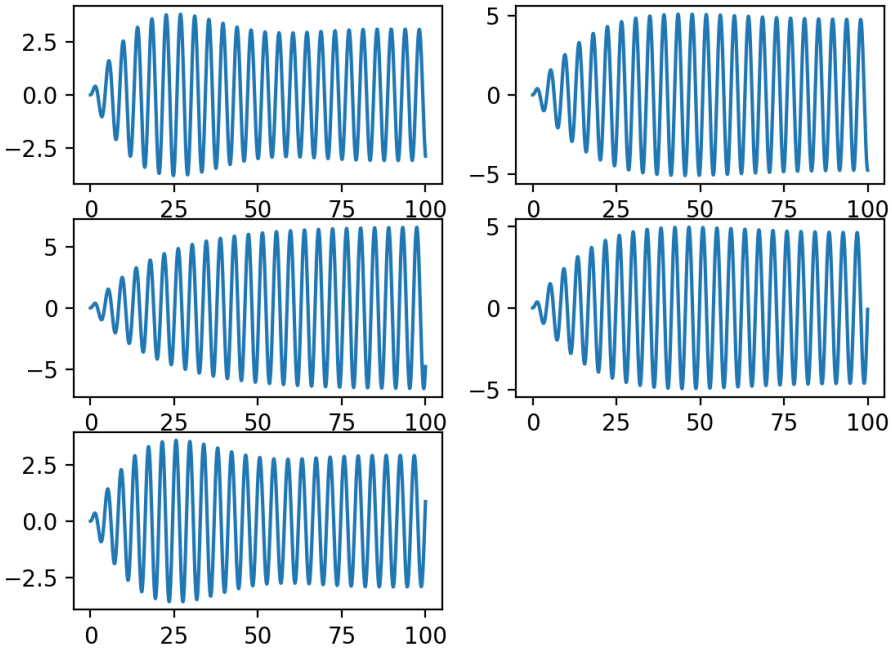

Figure 3: solution x(t) with ω in [1.4, 1.45, 1.5, 1.55, 1.6]

0.2.6 Observation

From the obtained result, we can clearly see that there is a settling of amplitude in frequencies other than 1.5. It is because natural frequency of the system is 1.5.

$$\ddot{x} + 2.25x = 0 \tag{3}$$

So,

$$\boxed{\omega_n = 1.5} \tag{4}$$

0.2.7 Q4

Given is a coupled spring system of following.

$$\ddot{x} + (x - y) = 0 \tag{5}$$

$$\ddot{y} + 2(y - x) = 0 \tag{6}$$

with initial conditions of x(0) = 1, $\dot{x}(0) = \dot{y}(0) = \text{y}(0) = 0$.
By substituting the y from first equation into second equation, we get,

$$\dddot{x} + 3\ddot{x} = 0 \tag{7}$$

By solving the equation in laplace domain, we get,

$$X(s) = \frac{s^2 + 2}{s^3 + 3s} \tag{8}$$

$$Y(s) = \frac{2}{s^3 + 3s} \tag{9}$$

```
def Q4(init_x:float=1, init_y:float=0, init_xdot:float=0, init_ydot:float=0):
    '''
    This function solves question 4
    '''
    # Given, x'' + x - y = 0 and y'' + 2y - 2x = 0
    # Using this, x''(0) = y(0) - x(0)
    # y''(0) = 2x(0) - 2y(0)
    # x'''(0) = y'(0) - x'(0)
    # Substituting eq1 in eq2, we get x'''' + 3x'' = 0
    # In laplace, s^4 . K - s^3 . x(0) - s^2 . x'(0) - s . x''(0) - x'''(0) + s^2 . K -

    init_xddot = init_y - init_x
    init_xdddot = init_ydot - init_xdot

    K_num = poly1d([init_x, init_xdot, 3*init_x+init_xddot, 3*init_xdot+init_xdddot])
    K_dec = poly1d([1, 0, 3, 0, 0])

    L_num = polyadd(polymul(K_num, [1, 0, 1]), polymul(K_dec, [-1*init_x, 0]))
    L_dec = K_dec

    K = sp.lti(K_num, K_dec)
    L = sp.lti(L_num, L_dec)

    _, k = sp.impulse(K, None, t)
    _, l = sp.impulse(L, None, t)

    plot(np.linspace(0, 20, N), k)
    xlabel(r'time $\longrightarrow$')
    ylabel(r'x(t) $\longrightarrow$')
    title('x(t) v/s t')
    show()

    plot(np.linspace(0, 20, N), l)
    xlabel(r'time $\longrightarrow$')
    ylabel(r'y(t) $\longrightarrow$')
    title('y(t) v/s t')
    show()
```

Converting them into time domain and plotting them, we get the following plots

0.2.8 Q5

Given an RLC circuit, we get the transfer function of H(s) as follows

$$H(s) = \frac{1}{1 + sRC + s^2LC} \tag{10}$$

Substituting the values of R, L, C and plotting the bode plots gives us the following

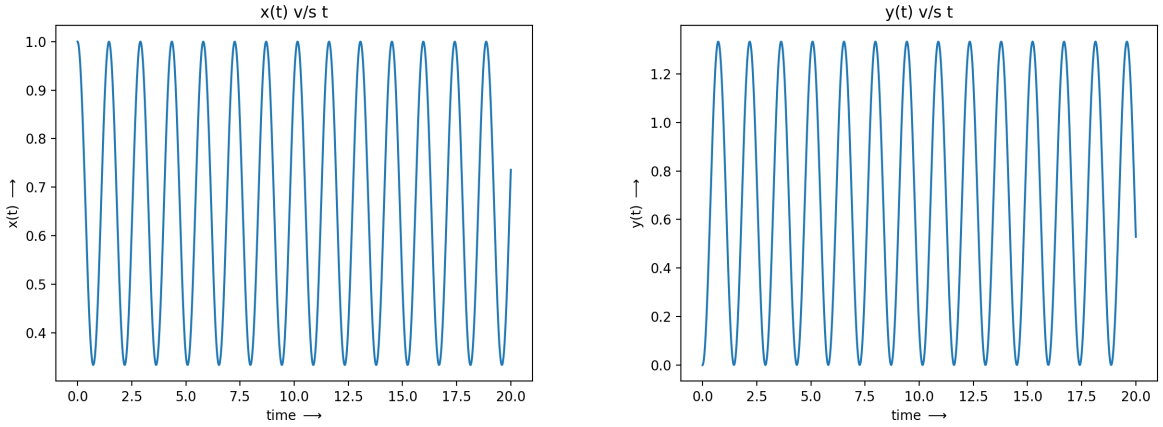

Figure 4: x(t) and y(t) respectively

```
def Q5(R:float=100, L:float=1e-6, C:float=1e-6):
    '''
    This function solves question 5
    '''
    # Transfer function is of the form 1/(1+sRC+s^2 LC)

    global T_num, T_dec, T

    T_num = poly1d([1])
    T_dec = poly1d([L*C, R*C, 1])

    T = sp.lti(T_num, T_dec)

    w, S, phi = T.bode()

    subplot(1, 2, 1)
    title('Magnitude Bode Plot of H(s)')
    xlabel('w(log)')
    ylabel(r'$\mid H(jw)\mid$ (log)')
    semilogx(w, S)
    subplot(1, 2, 2)
    title('Phase Bode Plot of H(s)')
    xlabel('w(log)')
    ylabel(r'$\angle H(jw)$ (log)')
    semilogx(w, phi)
    show()
```

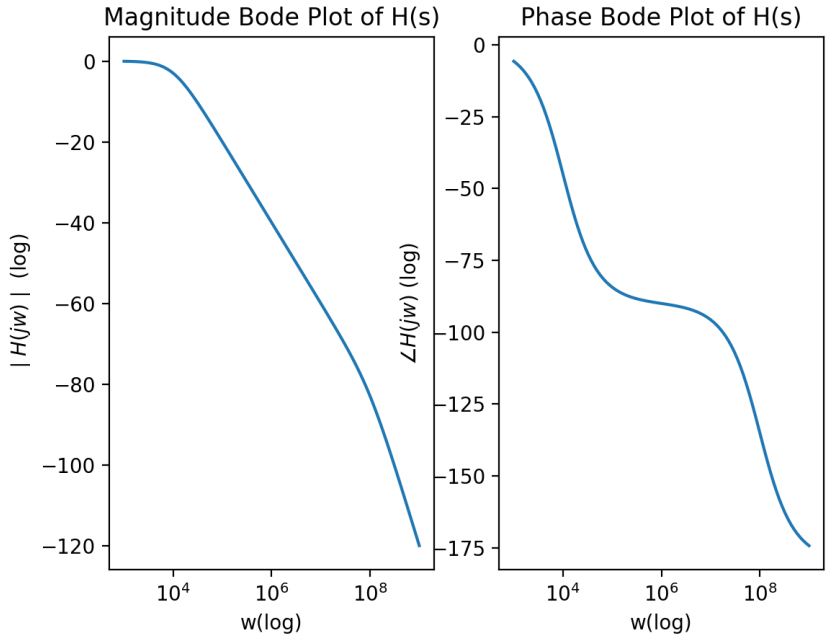

Figure 5: $\mid H(j\omega) \mid$ and $\angle H(j\omega)$ wrt w in semilogx scale

0.2.9 Q6

Given the transfer function H(s) from Q5, and the input voltage as follows

$$v_i(t) = cos(10^3 t)u(t) - cos(10^6 t)u(t) \tag{11}$$

Plotting the output voltage for two intervals of time,

- 0 to 30μs
- 0 to 10ms

We get the following graphs

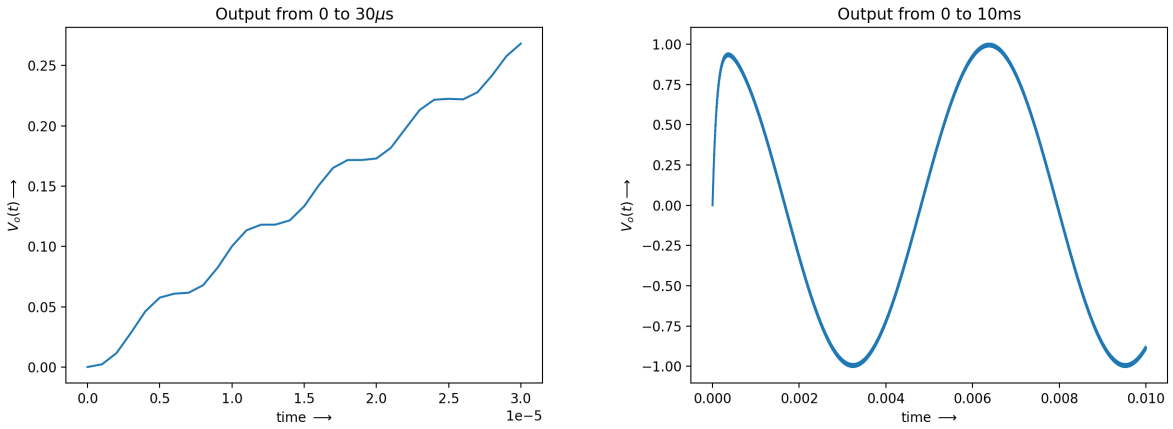

Figure 6: $v_o(t)$ at respective time intervals

```
def Q6():
    '''
    This function solves question 6
    '''
    Q5()
    t1 = np.arange(0, 30e-6, 1e-6)
    t2 = np.arange(0, 10e-3, 1e-6)

    _, vo1, _ = sp.lsim(T, vi(t1), t1)
    plot(t1, vo1)
    title(r'Output from 0 to 30$\mu$s')
    xlabel(r'time $\longrightarrow$')
    ylabel(r'${V_o}(t) \longrightarrow$')
    show()

    _, vo2, _ = sp.lsim(T, vi(t2), t2)
    plot(t2, vo2)
    title(r'Output from 0 to 10ms')
    xlabel(r'time $\longrightarrow$')
    ylabel(r'${V_o}(t) \longrightarrow$')
    show()
```

0.2.10 Observation

It is clear from the magnitude bode plot that higher frequency terms do not appear in the output as much as lower frequency terms. This system is called **Low Pass Filter**. So, $cos(10^6 t)$ component has lesser amplitude than $cos(10^3 t)$. When we try to see the output in time domain for the first time interval, it shows us many ripples which are climbing up. This is because the higher frequency component is dominating in shorter intervals. This argument can be supported by this image at a random small interval And as we look at the output in larger

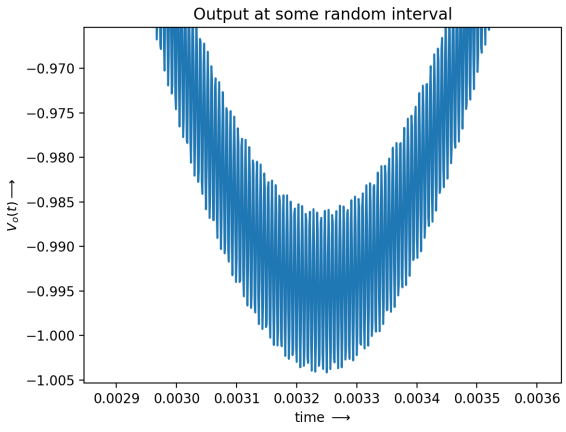

Figure 7: Output at some random interval

intervals, we see that $cos(10^3 t)$ component is dominating. This argument can be explained by the following image.

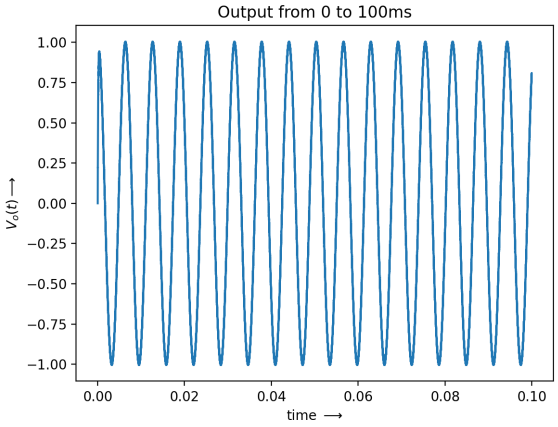

Figure 8: Output at some random interval

0.3 Takeaways

- I got to know how to deal with transfer functions in code
- I got to revise my newtonian mechanics concepts and solving the differential equations
- I got to see how actual bode plots are, instead of the approximate ones we draw on paper
- I got some clarity on how LPFs work and how they can be realised in a simpler way